```
class BinHex():
    masque8 = 0xFF
    masque16 = 0xFFFF
    masque32 = 0xFFFFFFFF
    masque64 = 0xFFFFFFFFFFFFFFFF
    fmtbin, fmthex = "0b", "0x"
    bases = ("bin", "hex")
    longueurs = (8, 16, 32, 64)

    def __init__(self, x: int) -> None:
        self._x = x

    def afficher(self, base: str = None, longueur: int = None) -> str:
        if base not in self.bases or base is None:
            raise AttributeError(f"Base '{base}' inconnue.")

        if longueur not in self.longueurs or longueur is None:
            raise AttributeError(f"Longueur '{longueur}' non gérée.")

        lng = ""
        fmt = self.fmtbin if base == "bin" else self.fmthex

        match longueur:
            case 8:
                lng = "08b" if base == "bin" else "02X"
                val = format(self._x & self.masque8, lng)
            case 16:
                lng = "016b" if base == "bin" else "04X"
                val = format(self._x & self.masque16, lng)
            case 32:
                lng = "032b" if base == "bin" else "08X"
                val = format(self._x & self.masque32, lng)
            case 64:
                lng = "064b" if base == "bin" else "016X"
                val = format(self._x & self.masque64, lng)

        return f"{fmt}{val}"


    def permutation(self, longueur: int = None) -> "BinHex":
        if longueur not in self.longueurs or longueur is None:
            raise TypeError(f"'{longueur}' non gérée.")

        match longueur:
            case 8:
                self._x = ((self._x & 0x0F) << 4) | ((self._x & 0xF0) >> 4)
            case 16:
                self._x = ((self._x & 0x00FF) << 8) | ((self._x & 0xFF00) >> 8)
            case 32:
                self._x = ((self._x & 0x000000FF) << 24) | ((self._x & 0x0000FF00) << 8) | \
                          ((self._x & 0x00FF0000) >> 8) | ((self._x & 0xFF000000) >> 24)
            case 64:
                self._x = ((self._x & 0x00000000000000FF) << 56) | ((self._x & 0x000000000000FF00) << 40) | \
                          ((self._x & 0x0000000000FF0000) << 24) | ((self._x & 0x00000000FF000000) << 8)  | \
                          ((self._x & 0x000000FF00000000) >> 8)  | ((self._x & 0x0000FF0000000000) >> 24) | \
                          ((self._x & 0x00FF000000000000) >> 40) | ((self._x & 0xFF00000000000000) >> 56)

        return self


    def mise_a_un_bit(self, bit: int) -> "BinHex":
        self._x |= (1 << bit)
        return self

    def mise_a_zero_bit(self, bit: int) -> "BinHex":
        self._x &= ~(1 << bit)
        return self

    def inverser_bit(self, bit: int) -> "BinHex":
        self._x ^= (1 << bit)
        return self

    def tester_bit(self, bit: int) -> bool:
        return ((self._x >> bit) & 1) == 1

    def puissance_deux(self) -> bool:
        return self._x & (self._x - 1) == 0

    def meme_signe(self, n: int) -> bool:
        return (self._x ^ n) > 0


if __name__ == "__main__":
    val8b = BinHex(0b11110000)
    val16b = BinHex(0xFACE)
    val32h = BinHex(0x00010000)
    val64h = BinHex(0x0123456789ABCDEF)

    separateur = "-" * 60
    print(f"Valeur 8 bits (bin) : {val8b.afficher("bin", 8)}")
    print(f"Valeur 16 bits (bin) : {val16b.afficher("bin", 16)}")
    print(f"Valeur 32 bits (bin) : {val32h.afficher("bin", 32)}")
    print(f"Valeur 64 bits (bin) : {val64h.afficher("bin", 64)}")
    print(separateur)
    print(f"Valeur 8 bits (hex) : {val8b.afficher("hex", 8)}")
    print(f"Valeur 16 bits (hex) : {val16b.afficher("hex", 16)}")
    print(f"Valeur 32 bits (hex) : {val32h.afficher("hex", 32)}")
    print(f"Valeur 64 bits (hex) : {val64h.afficher("hex", 64)}")
    print(separateur)
    print(f"Swap 8 bits : {val8b.permutation(8).afficher("bin", 8)}")
    print(f"Swap 16 bits : {val16b.permutation(16).afficher("bin", 16)}")
    print(f"Swap 32 bits : {val32h.permutation(32).afficher("bin", 32)}")
    print(f"Swap 64 bits : {val64h.permutation(64).afficher("bin", 64)}")
    print(separateur)
    print(f"Valeur 8 bits (hex) : {val8b.afficher("hex", 8)}")
    print(f"Valeur 16 bits (hex) : {val16b.afficher("hex", 16)}")
    print(f"Valeur 32 bits (hex) : {val32h.afficher("hex", 32)}")
    print(f"Valeur 64 bits (hex) : {val64h.afficher("hex", 64)}")
    print(separateur)
    print(f"Mise à zéro du bit 2 et mise : {val8b.mise_a_zero_bit(2).afficher("bin", 8)}")
    print(f"Mise à un du bit 7 : {val8b.mise_a_un_bit(7).afficher("bin", 8)}")
    print(f"31e bit de val32h à 1 ? {val32h.tester_bit(31)}")
    print(f"Inversion du bit 62 : {val64h.inverser_bit(62).afficher("bin", 64)}")
    print(separateur)
    print(f"Est-ce que 'val16b' est une puissance de 2 ? {val32h.puissance_deux()}")
    print(f"Est-ce que 'val32h' et -1023409 sont de même signe ? {val32h.meme_signe(1023409)}")


    # si besoin d'afficher en base 10 (décimale)
    # import ast
    # print(ast.literal_eval(val8b.afficher("hex", 8)))
    # print(ast.literal_eval(val16b.afficher("hex", 16)))
    # print(ast.literal_eval(val32h.afficher("hex", 32)))
    # print(ast.literal_eval(val64h.afficher("hex", 64)))
```